StructADMM: Achieving Ultrahigh Efficiency in Structured Pruning for DNNs

Tianyun Zhang, Shaokai Ye, Xiaoyu Feng, *Graduate Student Member, IEEE*, Xiaolong Ma, Kaiqi Zhang, Zhengang Li, Jian Tang, *Fellow, IEEE*, Sijia Liu, *Member, IEEE*, Xue Lin, *Member, IEEE*, Yongpan Liu, *Senior Member, IEEE*, Makan Fardad, and Yanzhi Wang, *Member, IEEE*

***Abstract*— Weight pruning methods of deep neural networks (DNNs) have been demonstrated to achieve a good model pruning rate without loss of accuracy, thereby alleviating the significant computation/storage requirements of large-scale DNNs. Structured weight pruning methods have been proposed to overcome the limitation of irregular network structure and demonstrated actual GPU acceleration. However, in prior work, the pruning rate (degree of sparsity) and GPU acceleration are limited (to less than 50%) when accuracy needs to be maintained. In this work, we overcome these limitations by proposing a unified, systematic framework of structured weight pruning for DNNs. It is a framework that can be used to induce different types of structured sparsity, such as filterwise, channelwise, and shapewise sparsity, as well as nonstructured sparsity. The proposed framework incorporates stochastic gradient descent (SGD; or ADAM) with alternating direction method of multipliers (ADMM) and can be understood as a dynamic regularization method in which the regularization target is analytically updated in each iteration. Leveraging special characteristics of ADMM, we further propose a progressive, multistep weight pruning framework and a network purification and unused path removal procedure, in order to achieve higher pruning rate without accuracy loss. Without loss of accuracy on the AlexNet model, we achieve 2.58× and 3.65× average measured speedup on two GPUs, clearly outperforming the prior work. The average speedups reach 3.15× and 8.52× when allowing a moderate accuracy loss of 2%. In this case, the model compression for convolutional layers is 15.0×, corresponding to 11.93× measured CPU speedup. As another example, for the ResNet-18 model on the CIFAR-10 data set, we achieve an unprecedented 54.2× structured pruning rate on CONV layers. This is 32× higher pruning rate compared with recent work and can further translate into 7.6× inference time speedup on the Adreno 640 mobile GPU compared with the original, unpruned DNN model. We share our codes and models at the link http://bit.ly/2M0V7DO.**

***Index Terms*— Alternating direction method of multipliers (ADMM), deep neural networks (DNNs), hardware acceleration, weight pruning.**

I. Introduction

DEEP neural networks (DNNs) utilize multiple functional layers cascaded together to extract features at multiple levels of abstraction [1]–[6] and are thus both computationally and storage intensive. As a result, many studies on DNN model compression are underway, including weight pruning [7]–[11], low-rank approximation [12]–[14], and low displacement rank approximation (structured matrices) [15]–[17]. Weight pruning can achieve a high model pruning rate without loss of accuracy. An early work [7], [8] adopts an iterative weight pruning heuristic and results in a sparse neural network structure. It can achieve 9× weight reduction with no accuracy loss on AlexNet [1]. This weight pruning method has been extended in [9], [11], and [18]–[23] to either use more sophisticated algorithms to achieve a higher weight pruning rate or to obtain a fine-grained tradeoff between a higher pruning rate and a lower accuracy degradation.

Despite the promising results, these general weight pruning methods often produce nonstructured and irregular connectivity in DNNs. This leads to degradation in the degree of parallelism and actual performance in GPU and hardware platforms. Moreover, the weight pruning rate is mainly achieved through compressing the fully connected (FC) layers [7], [8], [18], which are less computationally intensive compared with convolutional (CONV) layers and are becoming less important in state-of-the-art DNNs such as ResNet [24]. To address these limitations, recent work [10], [25] have proposed to learn structured sparsity, including sparsity at the levels of filters, channels, filter shapes, and layer depth. These works focus on CONV layers and actual GPU speedup is reported as a result of structured sparsity [10]. However, these structured weight pruning methods are based on fixed regularization techniques and are still quite heuristic [10], [25]. The weight pruning rate and GPU acceleration are both quite limited. For example, the average weight

Manuscript received March 24, 2020; revised August 12, 2020; accepted November 24, 2020. This work was supported by the National Science Foundation under Award CAREER CMMI-1750531, Award ECCS-1609916, Award CCF-1919117, Award CNS-1909172, and Award CNS-1704662. *(Tianyun Zhang and Shaokai Ye contributed equally to this work.) (Corresponding author: Yanzhi Wang.)*

Tianyun Zhang, Kaiqi Zhang, Jian Tang, and Makan Fardad are with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244 USA.

Shaokai Ye, Xiaoyu Feng, and Yongpan Liu are with the Department of Electronic Engineering, Tsinghua University, Beijing 100084, China.

Xiaolong Ma, Zhengang Li, Xue Lin, and Yanzhi Wang are with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 USA (e-mail: yanz.wang@northeastern.edu).

Sijia Liu is with the IBM Cambridge Research Center, Cambridge, MA 02141 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2020.3045153.

Digital Object Identifier 10.1109/TNNLS.2020.3045153

pruning rate on CONV layers of AlexNet is only 1.5× without any accuracy loss, corresponding to 33.3% sparsity.

In this work, we overcome this limitation by proposing a unified, systematic framework of structured weight pruning for DNNs, named StructADMM, based on the powerful optimization tool alternating direction method of multipliers (ADMM) [26], [27] shown to perform well on combinatorial constraints. It is a unified framework for different types of structured sparsity, such as filterwise, channelwise, and shapewise sparsity, as well as nonstructured sparsity. It is a systematic framework of dynamic ADMM regularization and masked mapping and retraining steps, guaranteeing solution feasibility (satisfying all constraints) and providing high solution quality. It achieves a significant improvement in weight pruning rate under the same accuracy, along with fast convergence rate. In the context of deep learning, the StructADMM framework can be understood as a smart and dynamic regularization technique in which the regularization target is analytically updated in each iteration.

Beyond the above single-step, one-shot ADMM framework, we observe the opportunity of performing further weight pruning from the results. This is due to the special property of L_2-based ADMM regularization process. This observation suggests a progressive, multistep model compression framework using ADMM. In the progressive framework, the pruning result from the previous step serves as an intermediate result and starting point for the subsequent step. It has an additional benefit of reducing the search space for (structured) weight pruning within each step. The detailed procedure and hyperparameter determination process have been carefully designed toward ultrahigh weight pruning rates.

During the postprocessing procedure, we find that after model retraining, some weights become less contributing to the network performance. To take advantage of this characteristics, we propose a novel algorithm to detect and remove the redundant weights that slip away from ADMM (structured) weight pruning. Also, we are the first to discover the unused path in a structured pruned DNN model and design an effective optimization framework to further boost compression rate as well as maintain high network accuracy.

We conduct extensive experiments on StructADMM framework using the ImageNet, CIFAR-10, MNIST, and UCF-101 data sets, using AlexNet, VGGNet, ResNet-18/ResNet-50, MobileNet-V2, and LeNet-5 DNN models. For acceleration evaluation, we have tested on two NVIDIA GPUs (NVIDIA 1080Ti and Jetson TX2), Intel i7-6700K CPU, as well as the Qualcomm Adreno 640 mobile GPU in Samsung Galaxy S10 smartphone. We have validated significant benefits of StructADMM in both weight pruning rates and actual system accelerations, with respect to the original DNN model and a large collection of prior work. For example, without accuracy loss on the AlexNet model (with over 2% accuracy enhancement in baseline accuracy), we achieve 2.58× and 3.65× average measured speedup on two GPUs, which clearly outperforms the GPU acceleration of 49% reported in SSL [10]. The speedups reach 3.15× and 8.52× when allowing a moderate accuracy loss of 2%. In this case, the model compression for CONV layers is 15.0×, corresponding to 11.93× CPU speedup. As another example, for the ResNet-18 model on the CIFAR-10 data set, we achieve an unprecedented 54.2× structured pruning rate on CONV layers when a two-step, progressive framework is utilized. This is 32× higher pruning rate compared with recent work or over 100× fewer number of actual weight parameters considering the difference of original DNN. It can further translate into 7.6× inference time speedup on the Adreno 640 mobile GPU compared with the original, unpruned DNN model.

An early version of this work appeared in [28]. In this article, we expand our contributions on the following aspects.

1) Our early version [28] only focus on nonstructured pruning and evaluates the performance on LeNet-5 and AlexNet. In this article, we propose a unified framework for different types of structured sparsity such as filterwise, channelwise, and shapewise sparsity, as well as nonstructured sparsity. Also, we evaluate the performance on a variety of models, such as VGG-16, ResNet, and MobileNet. Also, we focus on the actual inference time speedup in various platforms achieved by our structured pruning method.
2) We propose a progressive, multistep framework for DNN compression, which can be used for both structured pruning and nonstructured pruning. It notably improves the pruning rate, e.g., it increases the pruning rate on AlexNet from 21× achieved by single-step ADMM [28] to 31×.
3) We propose network purification and unused path removal step for structured pruning, which further increases the pruning rate without accuracy loss.

The rest of this article is organized as follows. Section II discusses the related works. Section III briefly introduces the background of ADMM. The proposed unified framework for DNN weight pruning is investigated in Section IV. Section V illustrates our proposed methods to increase the pruning rate. Section VI discusses how to set hyperparameters and the effectiveness of our proposed framework. We compare the experiment results of our proposed method with state-of-the-art weight pruning methods for structured and nonstructured pruning in Sections VII and VIII, respectively. Finally, Section IX concludes this article.

II. Related Work

A. General, Nonstructured Weight Pruning

The early work by Han *et al.* [7], [8] achieved 9× reduction in the number of parameters in AlexNet and 13× in VGG-16. However, most reduction is achieved in FC layers, and 2.7× reduction achieved in CONV layers will not lead to an overall acceleration in GPU [10]. Extensions of iterative weight pruning, such as [18] (dynamic network surgery), [9] (NeST), and [29], use more delicate algorithms such as selective weight growing and pruning. However, the weight pruning rates on CONV layers are still limited, e.g., 3.1× in [18], 3.23× in [9], and 4.16× in [29] for AlexNet with no accuracy degradation. This level of nonstructured weight pruning cannot guarantee GPU acceleration. In fact, our StructADMM framework can achieve 16.1× nonstructured weight pruning in CONV layers

of AlexNet without accuracy degradation; however, only minor GPU acceleration is actually observed.

B. Structured Weight Pruning

To overcome the limitation in nonstructured, irregular weight pruning, SSL [10] proposes to learn structured sparsity at the levels of filters, channels, filter shapes, layer depth, and so on. This work is one of the first with actually measured GPU accelerations. This is because CONV layers after structured pruning will transform to a full matrix multiplication in GPU (with reduced matrix size). However, the weight pruning rate and GPU acceleration are both limited. The average weight pruning rate on CONV layers of AlexNet is only 1.5× without accuracy loss. The reported GPU acceleration is 49%. Besides, another work [25] achieves 2× channel pruning with 1% accuracy degradation on VGGNet.

C. Other Types of DNN Model Compression Techniques

There are many other types of DNN model compression techniques. Weight quantization leverages the inherent redundancy in the number of bits for weight representation. Many of the prior works [30]–[36] are directed at quantization of weights to binary values, ternary values, or powers of 2 to facilitate hardware implementations, with acceptable accuracy loss. The state-of-the-art techniques [37], [38] adopt an iterative quantization and retraining framework, with some degree of randomness incorporated into the quantization step. This method results in less than 3% accuracy loss on AlexNet for binary weight quantization [38]. Furthermore, knowledge distillation leverages the idea that a smaller student model can absorb knowledge from the larger teacher model [39]–[42], low-rank approximation using single-value decomposition (SVD) [12]–[14], and low-displacement rank approximation using structured matrices, such as circulant matrices [15], [43] and Toeplitz matrices [16], [17]. These techniques result in a regular network structure, but in general a lower pruning rate and larger accuracy degradation compared with parameter pruning. We point out that these compression techniques are compatible with ADMM and will be the topic of future investigations orthogonal to this work.

Besides the works we mention above, there are also several representative recent works in this field. Yu *et al.* [44], Li *et al.* [45], and He *et al.* [46] defined metrics to measure the importance of the weights, and they prune the weights that are less important according to their metrics. Zhuang *et al.* [47] and Yang *et al.* [48] defined optimization targets to generate sparse DNNs, and they set the optimization target as a regularization term when they train the DNNs. In these methods, the authors decide the importance of weights on a static model or setting a static optimization target as the regularization term. However, in our method, the regularization targets are updated dynamically during the training procedure. This is the major difference between our method and these methods. Note that there are also recent works use dynamically updated approaches in the training to prune DNNs, such as C-SGD [49] and CNN-FCF [50]. C-SGD trains several filters to collapse into a single point in the parameter hyperspace and then remove the identical filters. CNN-FCF defines dynamically updated binary scalars to constraint the filters and remove the filters corresponding to 0-valued scalars after convergence. Both of the two methods focus only on filter pruning on DNNs, but our framework is unified for different kinds of structured pruning, as well as nonstructured pruning.

III. Background of ADMM

ADMM was first introduced in the 1970s, and theoretical results in the following decades are collected in [26]. It is a powerful method for solving regularized convex optimization problems, especially for problems in applied statistics and machine learning. Moreover, recent works [51], [52] demonstrate that ADMM is also a good tool for solving nonconvex problems, potentially with combinatorial constraints since it can converge to a solution that may not be globally optimal but is sufficiently good for many applications.

ADMM is based on decomposing an optimization problem, which is difficult to solve directly, into two subproblems that can be solved separately and efficiently. For example, the optimization problem

$$\min_{\mathbf{x}} \; f(\mathbf{x}) + g(\mathbf{x}) \tag{1}$$

lends itself well to the application of ADMM if $f(\cdot)$ is differentiable and $g(\cdot)$ is nondifferentiable but has some structure that can be exploited. Common instances of g are the ℓ_1 norm and the indicator function of a constraint set. To prepare it for the application of ADMM, the above problem is first rewritten as

$$\begin{aligned} &\min_{\mathbf{x}} \; f(\mathbf{x}) + g(\mathbf{z}) \\ &\text{s.t. } \mathbf{x} = \mathbf{z}. \end{aligned}$$

Next, via the introduction of the augmented Lagrangian, the above optimization problem can be decomposed into two subproblems in $\mathbf{x}$ and $\mathbf{z}$ [26]. The first subproblem is $\min_{\mathbf{x}} \; f(\mathbf{x}) + q_1(\mathbf{x})$, where $q_1(\cdot)$ is a quadratic function of its argument. Since q_1 is differentiable and convex, the complexity of solving this problem (e.g., via gradient descent) is the same as that of minimizing f. The second subproblem is $\min_{\mathbf{z}} \; g(\mathbf{z}) + q_2(\mathbf{z})$, where $q_2(\cdot)$ is a quadratic function of its argument. In problems where g has some special sturcture, for instance, if it is a regularizer in (1), then exploiting the properties of g allows this problem to be solved analytically. More details regarding the application of ADMM to the weight pruning problem will be demonstrated in Section IV.

IV. Unified StructADMM Framework for Structured Pruning

A. Problem Formulation of Structured Pruning

Consider an N-layer DNN in which the first M layers are CONV layers and the rest are FC layers. The weights and biases of the ith layer are, respectively, denoted by $\mathbf{W}_i$ and $\mathbf{b}_i$. Assume that the input to the DNN is $\mathbf{x}$. Every column of $\mathbf{x}$ corresponds to a training image, and the number t of columns determines the number of training images in the input batch. The input $\mathbf{x}$ will enter the first layer and the output of the first layer is calculated by

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

where $\mathbf{h}_1$ and $\mathbf{b}_1$ have t columns and $\mathbf{b}_1$ is a matrix with identical columns. The nonlinear activation function $\sigma(\cdot)$ acts entrywise on its argument and is typically chosen to be the ReLU function [53] in state-of-the-art DNNs. Since the output of one layer is the input of the next, the output of the ith layer for $i = 2, \ldots, N-1$ is given by

$$\mathbf{h}_i = \sigma(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i).$$

The output of the DNN corresponding to a batch of images is

$$\mathbf{s} = \mathbf{W}_N \mathbf{h}_{N-1} + \mathbf{b}_N.$$

In this case, $\mathbf{s}$ is a $k \times t$ matrix, where k is the number of classes in the classification and t is the number of training images in the batch. The element $\mathbf{s}_{ij}$ in matrix $\mathbf{s}$ is the score of the jth training image corresponding to the ith class. The total loss of the DNN is calculated as

$$f(\{\mathbf{W}_1, \ldots, \mathbf{W}_N\}, \{\mathbf{b}_1, \ldots, \mathbf{b}_N\}) = -\frac{1}{t}\sum_{j=1}^{t} \log \frac{e^{\mathbf{s}_{y_j j}}}{\sum_{i=1}^{k} e^{\mathbf{s}_{ij}}} + \lambda \sum_{i=1}^{N} \|\mathbf{W}_i\|_F^2$$

where the first term is cross-entropy loss, y_j is the correct class of the jth image, and the second term is L_2 weight regularization.

Hereafter, for simplicity of notation, we write $\{\mathbf{W}_i\}_{i=1}^N$, or simply $\{\mathbf{W}_i\}$ instead of $\{\mathbf{W}_1, \ldots, \mathbf{W}_N\}$. The same notational convention applies other variables or parameters. The training of a DNN is a process of minimizing the loss by updating weights and biases. If we use the gradient descent method, then the update at every step is

$$\mathbf{W}_i = \mathbf{W}_i - \alpha \frac{\partial f\left(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\right)}{\partial \mathbf{W}_i}$$

$$\mathbf{b}_i = \mathbf{b}_i - \alpha \frac{\partial f\left(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\right)}{\partial \mathbf{b}_i}$$

computed for $i = 1, \ldots, N$, where α is the learning rate.

In this article, our objective is to implement structured pruning on DNNs. In the following discussion, we focus on the CONV layers because they have the highest computation requirements. More specifically, we minimize the loss function subject to specific structured sparsity constraints on the weights in the CONV layers, i.e.,

$$\min_{\{\mathbf{W}_i\},\{\mathbf{b}_i\}} \left(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\right) \quad \text{s.t. } \mathbf{W}_i \in \mathbf{S}_i, \quad i = 1, \ldots, M \tag{2}$$

where $\mathbf{S}_i$ is the set of $\mathbf{W}_i$ with a specific "structure." Next, we introduce constraint sets corresponding to different types of structured sparsity. Nonstructured, irregular sparsity is also included in the framework. The suitability for GPU acceleration is discussed for different types of sparsity, and we finally introduce the proper combination of structured sparsity to facilitate GPU accelerations. The details of different types of structures will be discussed later in Section IV-C.

In problem (2), the constraint is nonconvex and combinatorial. As a result, this problem cannot be solved directly by stochastic gradient descent (SGD) methods [54] (or ADAM [55]). However, the property that $\mathbf{W}_i$ satisfies certain combinatorial "structures" allows us to integrate the ADMM framework with SGD to effectively solve this problem.

B. Proposed StructADMM Framework

To apply the ADMM framework, we define indicator functions to incorporate combinatorial constraints into the objective function and define auxiliary variables that allow us to decompose the optimization problem into two subproblems that individually can be solved effectively. In what follows, we elaborate on these steps.

Corresponding to every set $\mathbf{S}_i$, $i = 1, \ldots, M$, we define the indicator function

$$g_i(\mathbf{W}_i) = \begin{cases} 0, & \text{if } \mathbf{W}_i \in \mathbf{S}_i \\ +\infty, & \text{otherwise.} \end{cases}$$

Furthermore, we incorporate auxiliary variables $\mathbf{Z}_i$, $i = 1, \ldots, M$ with the restriction that $\mathbf{Z}_i = \mathbf{W}_i$. The original problem (2) is then equivalent to

$$\min_{\{\mathbf{W}_i\},\{\mathbf{b}_i\}} f\left(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\right) + \sum_{i=1}^{M} g_i(\mathbf{Z}_i) \quad \text{s.t. } \mathbf{W}_i = \mathbf{Z}_i, \quad i = 1, \ldots, M. \tag{3}$$

The augmented Lagrangian [26] of problem (3) is defined by

$$L_\rho\left(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N, \{\mathbf{Z}_i\}_{i=1}^M, \{\Lambda_i\}_{i=1}^M\right) = f\left(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\right) + \sum_{i=1}^{M} g_i(\mathbf{Z}_i) + \sum_{i=1}^{M} \mathrm{tr}[\Lambda_i^T(\mathbf{W}_i - \mathbf{Z}_i)] + \sum_{i=1}^{M} \frac{\rho_i}{2}\|\mathbf{W}_i - \mathbf{Z}_i\|_F^2$$

where $\|\cdot\|_F$ denotes the Frobenius norm, $\{\Lambda_i\}_{i=1}^M$ are the dual variables, and the penalty parameters $\{\rho_i\}_{i=1}^M$ are positive. With the scaled dual variable $\mathbf{U}_i = (1/\rho_i)\Lambda_i$ for $i = 1, \ldots, M$, the augmented Lagrangian can be equivalently rewritten as

$$L_\rho\left(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N, \{\mathbf{Z}_i\}_{i=1}^M, \{\Lambda_i\}_{i=1}^M\right) = f\left(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\right) \sum_{i=1}^{M} g_i(\mathbf{Z}_i) + \sum_{i=1}^{M} \frac{\rho_i}{2}\|\mathbf{W}_i - \mathbf{Z}_i + \mathbf{U}_i\|_F^2 - \sum_{i=1}^{M} \frac{\rho_i}{2}\|\mathbf{U}_i\|_F^2.$$

ADMM consists the following iterations for $k = 0, 1, \ldots,$ [26], [56]:

$$\left\{\mathbf{W}_i^{k+1}\right\}_{i=1}^N, \left\{\mathbf{b}_i^{k+1}\right\}_{i=1}^N := \arg\min_{\{\mathbf{W}_i\},\{\mathbf{b}_i\}} L_\rho\left(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N, \{\mathbf{Z}_i^k\}_{i=1}^M, \{\mathbf{U}_i^k\}_{i=1}^M\right) \tag{4}$$

$$\left\{\mathbf{Z}_i^{k+1}\right\}_{i=1}^M := \arg\min_{\{\mathbf{Z}_i\}} L_\rho\left(\left\{\mathbf{W}_i^{k+1}\right\}_{i=1}^N, \left\{\mathbf{b}_i^{k+1}\right\}_{i=1}^N, \{\mathbf{Z}_i\}_{i=1}^M, \left\{\mathbf{U}_i^k\right\}_{i=1}^M\right) \tag{5}$$

$$\mathbf{U}_i^{k+1} := \mathbf{U}_i^k + \mathbf{W}_i^{k+1} - \mathbf{Z}_i^{k+1} \quad \text{for } i = 1, \ldots, M \tag{6}$$

until for $i = 1, \ldots, M$, both of the following conditions are satisfied:

$$\left\|\mathbf{W}_i^{k+1} - \mathbf{Z}_i^{k+1}\right\|_F^2 \le \epsilon_i, \quad \left\|\mathbf{Z}_i^{k+1} - \mathbf{Z}_i^k\right\|_F^2 \le \epsilon_i. \tag{7}$$

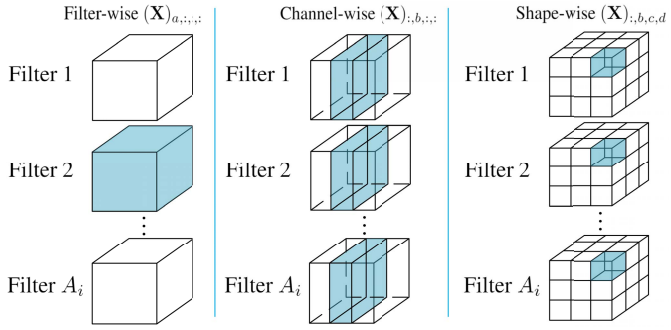


Fig. 1. Illustration of filterwise, channelwise, and shapewise structured sparsity from left to right.

In order to solve the overall pruning problem, we need to solve subproblems (4) and (5). More specifically, problem (4) can be formulated as

$$\min_{\{\mathbf{W}_i\},\{\mathbf{b}_i\}} f\left(\{\mathbf{W}_i\}_{i=1}^N,\{\mathbf{b}_i\}_{i=1}^N\right)+\sum_{i=1}^M \frac{\rho_i}{2}\left\|\mathbf{W}_i-\mathbf{Z}_i^k+\mathbf{U}_i^k\right\|_F^2 \tag{8}$$

where the first term in the objective function of (8) is the differentiable loss function of the DNN, and the second term is a quadratic regularization term of the $\mathbf{W}_i$'s, which is differentiable and convex. As a result, (8) can be solved by SGD. Although we cannot guarantee the global optimality of the solution, it is due to the nonconvexity of the DNN loss function rather than the quadratic term enrolled by our method.

On the other hand, problem (5) is given by

$$\min_{\{\mathbf{Z}_i\}} \sum_{i=1}^M g_i(\mathbf{Z}_i)+\sum_{i=1}^M \frac{\rho_i}{2}\left\|\mathbf{W}_i^{k+1}-\mathbf{Z}_i+\mathbf{U}_i^k\right\|_F^2. \tag{9}$$

Note that $g_i(\cdot)$ is the indicator function of $\mathbf{S}_i$, and thus, this subproblem can be solved analytically and optimally [26]. For $i=1,\ldots,M$, the optimal solution is

$$\mathbf{Z}_i^{k+1}=\Pi_{\mathbf{S}_i}\left(\mathbf{W}_i^{k+1}+\mathbf{U}_i^k\right) \tag{10}$$

where $\Pi_{\mathbf{S}_i}(\cdot)$ is the Euclidean projection of $\mathbf{W}_i^{k+1}+\mathbf{U}_i^k$ onto $\mathbf{S}_i$. The set $\mathbf{S}_i$ is different when we apply different types of structured sparsity. We will discuss how to implement the Euclidean projection to different types of structures in Section IV-C.

C. Solutions of Different Types of Structured Sparsity and Discussion

The collection of weights in the ith CONV layer is a 4-D tensor, i.e., $\mathbf{W}_i\in R^{A_i\times B_i\times C_i\times D_i}$, where A_i, B_i, C_i, and D_i are, respectively, the number of filters, the number of channels in a filter, the height of the filter, and the width of the filter, in layer i. In what follows, if $\mathbf{X}$ denotes the weight tensor in a specific layer, let $(\mathbf{X})_{a,:,:,:}$ denote the ath filter in $\mathbf{X}$, $(\mathbf{X})_{:,b,:,:}$ denote the bth channel, and $(\mathbf{X})_{:,b,c,d}$ denote the collection of weights located at position $(:,b,c,d)$ in every filter of $\mathbf{X}$, as shown in Fig. 1.

1) Filterwise Structured Sparsity: When we train a DNN with sparsity at the filter level, the constraint on the weights in the ith CONV layer is given by

$$\mathbf{W}_i\in\mathbf{S}_i:=\{\mathbf{X}\mid \text{ the number of nonzero filters in } \mathbf{X} \text{ is less than or equal to } \alpha_i\}.$$

Here, nonzero filter means that the filter contains some nonzero weight. To solve subproblem (10) with such constraints, we first calculate

$$O_a=\left\|\left(\mathbf{W}_i^{k+1}+\mathbf{U}_i^k\right)_{a,:,:,:}\right\|_F^2$$

for $a=1,\ldots,A_i$. We then keep α_i elements in $(\mathbf{W}_i^{k+1}+\mathbf{U}_i^k)_{a,:,:,:}$ corresponding to the α_i largest values in $\{O_a\}_{a=1}^{A_i}$ and set the rest to zero.

2) Channelwise Structured Sparsity: When we train a DNN with sparsity at the channel level, the constraint on the weights in the ith CONV layer is given by

$$\mathbf{W}_i\in\mathbf{S}_i:=\{\mathbf{X}\mid \text{ the number of nonzero channels in } \mathbf{X} \text{ is less than or equal to } \beta_i\}.$$

Here, we call the bth channel nonzero if $(\mathbf{X})_{:,b,:,:}$ contains some nonzero element. To solve subproblem (10) with such constraints, we first calculate

$$O_b=\left\|\left(\mathbf{W}_i^{k+1}+\mathbf{U}_i^k\right)_{:,b,:,:}\right\|_F^2$$

for $b=1,\ldots,B_i$. We then keep β_i elements in $(\mathbf{W}_i^{k+1}+\mathbf{U}_i^k)_{:,b,:,:}$ corresponding to the β_i largest values in $\{O_b\}_{b=1}^{B_i}$ and set the rest to zero.

3) Filter Shapewise Structured Sparsity: When we train a DNN with sparsity at the filter shape level, the constraint on the weights in the ith CONV layer is given by

$$\mathbf{W}_i\in\mathbf{S}_i:=\left\{\mathbf{X}\mid \text{the number of nonzero vectors in } \{\mathbf{X}_{:,b,c,d}\}_{b,c,d=1}^{B_i,C_i,D_i} \text{ is less than or equal to } \theta_i\right\}.$$

To solve subproblem (10) with such constraints, we first calculate

$$O_{b,c,d}=\left\|\left(\mathbf{W}_i^{k+1}+\mathbf{U}_i^k\right)_{:,b,c,d}\right\|_F^2$$

for $b=1,\ldots,B_i$, $c=1,\ldots,C_i$, and $d=1,\ldots,D_i$. We then keep θ_i elements in $(\mathbf{W}_i^{k+1}+\mathbf{U}_i^k)_{:,b,c,d}$ corresponding to the θ_i largest values in $\{O_{b,c,d}\}_{b,c,d=1}^{B_i,C_i,D_i}$ and set the rest to zero.

4) Nonstructured, Irregular Weight Sparsity: When we train a DNN with nonstructured weight sparsity, the constraint on the weights in the ith CONV layer is

$$\mathbf{W}_i\in\mathbf{S}_i:=\{\mathbf{X}\mid \text{ the number of nonzero elements in } \mathbf{X} \text{ is less than or equal to } \gamma_i\}.$$

To solve subproblem (10), we keep γ_i elements in $\mathbf{W}_i^{k+1}+\mathbf{U}_i^k$ with largest magnitudes and set the rest to zero.

5) Combination of Structured Sparsity to Facilitate GPU Acceleration: Convolutional computations in DNNs are commonly transformed to matrix multiplications by converting weight tensors and feature map tensors into matrices [57], named general matrix multiplication (GEMM). Filterwise sparsity corresponds to row pruning, whereas channelwise and filter shapewise sparsity correspond to column pruning in GEMM. The GEMM matrix maintains a full matrix with a number of rows/columns reduced, thereby enabling GPU acceleration. This is shown in Fig. 2, in which $W_{n,m,k}$ means the kth element in the mth channel of the nth filter. In the results, we will use row pruning to represent the results of filterwise sparsity in GEMM and use column pruning to represent the results of channelwise and filter shapewise sparsity.

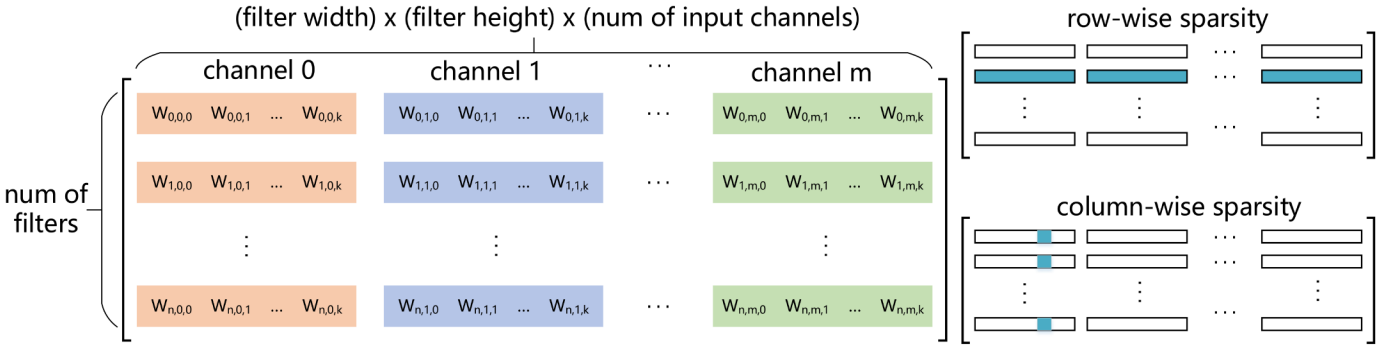


Fig. 2. Illustration of 2-D weight matrix for GEMM (left) and rowwise and columnwise sparsity (right).

D. Masked Retraining Step

For very small values of ϵ_i in (7), ADMM needs a large number of iterations to converge. However, in many applications, such as the weight pruning problem considered here, a slight increase in the value of ϵ_i can result in a significant speedup in convergence. On the other hand, when ADMM stops early, the weights to be pruned may not be identically zero, in the sense that there will be small nonzero elements contained in $\mathbf{W}_i$. To deal with this issue, we first perform the Euclidean projection to guarantee that the structured pruning constraints are satisfied. Next, we mask the zero weights and retrain the DNN with nonzero weights using training sets (while keeping the masked weights 0). In this way, test accuracy (solution quality) can be partially restored. Note that the convergence is much faster than training the original DNN since the starting point of the retraining is already close to the point, which can achieve the original test/validation accuracy.

E. Overall Illustration of Our Proposed Framework

We take the weight distribution of every (convolutional or FC) layer on LeNet-5 as an example to illustrate our systematic weight pruning method. The weight distributions at different stages are shown in Fig. 3. The subfigures in the left column show the weight distributions of the pretrained model, which serves as our starting point. The subfigures in the middle column show that after the convergence of ADMM for moderate values of ϵ_i, we observe a clear separation between weights whose values are close to zero and the remaining weights. To prune the weights rigorously, we set the values of the close-to-zero weights exactly to zero and retrain the DNN without updating these values. The subfigures in the right column show the weight distributions after our final retraining step. We observe that most of the weights are zero in every layer. This concludes our weight pruning procedure.

As mentioned before, the computation time for the ADMM procedure is similar to the training of the original DNN, and the single retraining step converges much faster than the original training. Consequently, the total computation time of our method is less than training the original DNN twice, which is much faster than the iterative pruning and training method in [7]. An overall illustration of our proposed StructADMM framework is shown in Algorithm 1, and the initialization of $\{\mathbf{Z}_i^0\}$ and $\{\mathbf{U}_i^0\}$ will be discussed in Section VI-A.

Algorithm 1 Overall Illustration of Our Proposed StructADMM Framework

Input: Pretrained model
Initialize $\{\mathbf{Z}_i^0\}$ and $\{\mathbf{U}_i^0\}$
Set $j = 0$ and $k = 0$.
Set T as the number of iterations of ADMM
Set β as the number of epochs in every iteration of ADMM
for $k \leq T$ **do**
  for $j \leq \beta$ **do**
    Solve problem (8) using one epoch of SGD or ADAM
  end for
  Update $\{\mathbf{Z}_i^{k+1}\}$ according to (10)
  Update $\{\mathbf{U}_i^{k+1}\}$ according to (6)
  if Condition (7) is satisfied **then**
    Break for loop
  end if
end for
Perform the Euclidean projection according to (10) to guarantee that the structured pruning constraints are satisfied. Then mask the zero weights and retrain the DNN with the non-zero weights.

V. Methods to Improve Pruning Rate

A. Progressive DNN Weight Pruning

The first motivation of the progressive framework is that during the implementation of the single-step weight pruning framework, we observe that there are a number of unpruned weights with values very close to zero. The reason is the L_2 regularization nature in the ADMM regularization step, which tends to generate very small, nonzero weight values even when they are not pruned. As the remaining number of nonzero weights is already significantly reduced during weight pruning, simply mapping these small-value weights to zero will result in accuracy degradation. On the other hand, this motivates us to perform weight pruning in a multistep, progressive manner. The weights that have been pruned in the previous step will be masked and only the remaining, nonzero weights will be considered in the subsequent step.

The second motivation of the progressive framework is to reduce the search space for weight pruning within each step. After all, weight pruning problems are essentially combinatorial optimizations. Although recently demonstrated to

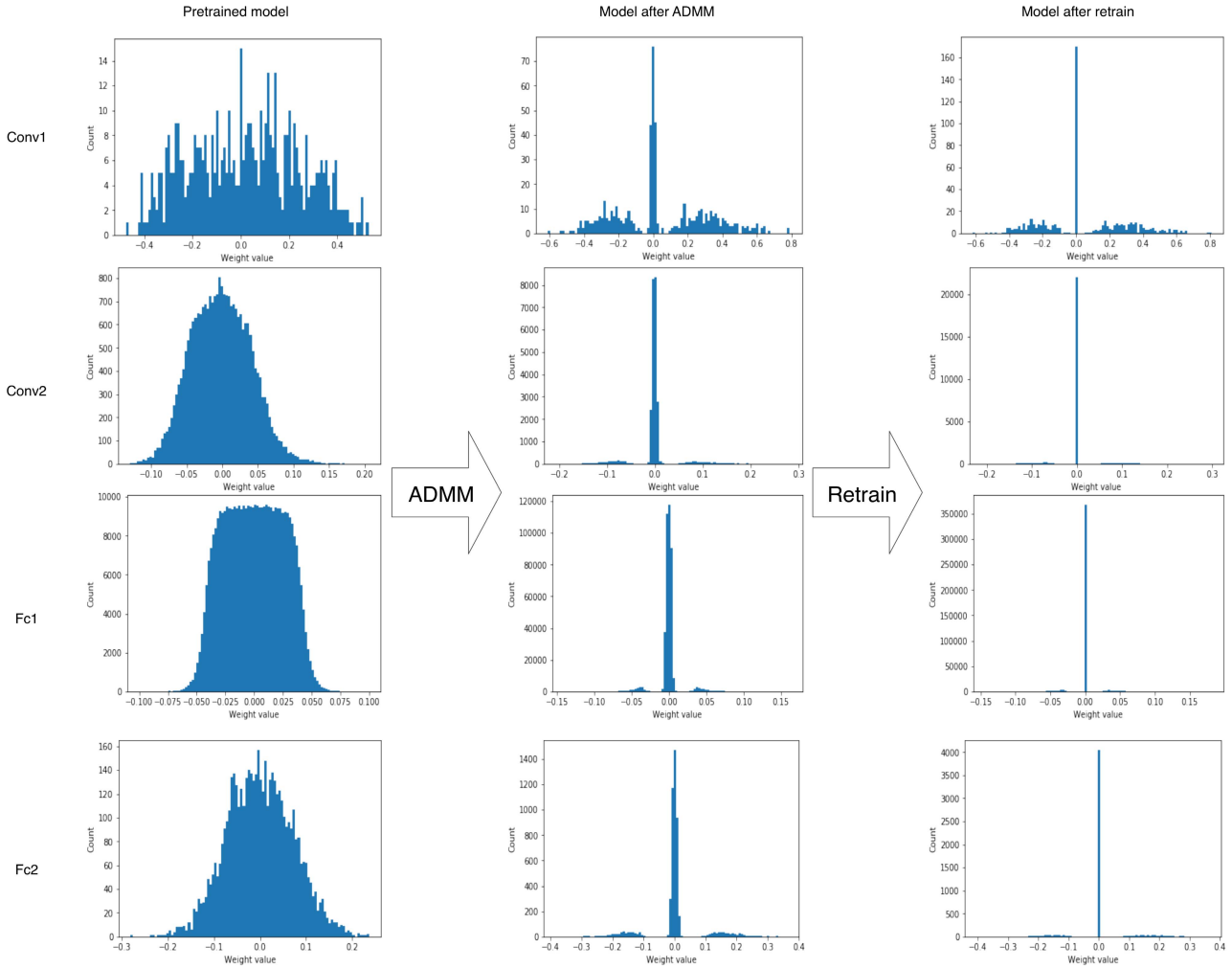


Fig. 3. Weight distribution of every (convolutional or FC) layer on LeNet-5. The subfigures in the left column are the weight distributions of the pretrained DNN model (serving as our starting point); the subfigures of the middle column are the weight distributions after the ADMM procedure; the subfigures of the right column are the weight distributions after our final retraining step. Note that the subfigures in the last column include a small number of nonzero weights that are not clearly visible due to the large number of zero weights.

generate superior results on this type of problems [58], [59], ADMM-based solution still has a superlinear increase of computational complexity as a function of solution space. As a result, the complexity becomes very high with ultrahigh compression rates (i.e., very large search space) beyond what can be achieved in prior work. The progressive framework, on the other hand, can mitigate this limitation and reduce the total training time (to $2\times$ or slightly higher than the training time of the original DNN).

A similar approach that masking the zero weights in the model generated from the previous iteration of pruning has been applied in [7] and [21], but our motivation is different from these works. The magnitude-based pruning method is used in [7] and [21], this method is heuristic, and the pruning rate has to be iteratively increased to avoid accuracy loss. One step of our ADMM-based method can achieve a much higher pruning rate than the iterative magnitude-based pruning method without accuracy loss. Also, our major purpose of using a progressive method is to reduce the search space in each step to achieve an ultrahigh pruning rate.

Also, masking the zero weights is not necessary for our method to reduce the search space. In our experiment, we find that even if we do not mask the zero weights pruned in each step when we start a new step, the value of the unimportant weights still keeps close to zero. This is because the ADMM-based regularization term prevents these unimportant weights to move away from zero. In conclusion, if we start from a model that is already pruned, the search space for our ADMM-based method is being reduced no matter if we mask the zero weights or not. We choose to mask the zero weights to make the training more rigorous.

Fig. 4 shows our proposed progressive DNN weight pruning method on the StructADMM framework. The single-step ADMM-based weight pruning is performed multiple times, each as a step in the progressive framework. The pruning results from the previous step serve as intermediate results and starting point for the subsequent step. Through extensive investigations, we conclude that a two-step progressive procedure will be in general sufficient for weight pruning, in which each step requires approximately the same number of

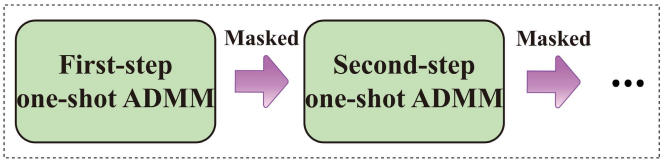


Fig. 4. Illustration of our proposed progressive DNN weight pruning method on the StructADMM framework.

training epochs as original DNN training. Further increase in the number of steps or the number of epochs in each step will result in only marginal improvement in the overall solution quality (e.g., 0.1%–0.2% accuracy improvement).

B. Network Purification and Unused Path Removal

After ADMM-based structured weight pruning, we propose the network purification and unused path removal step for further weight reduction without accuracy loss. First, as also noticed by prior work [25], a specific filter in layer i is responsible for generating one channel in layer $i+1$. As a result, removing the filter in layer i (in fact removing the batch norm results) also results in the removal of the corresponding channel, thereby achieving further weight reduction. Besides this straightforward procedure, there is a further margin of weight reduction based on the characteristics of ADMM regularization. As ADMM regularization is essentially a dynamic, L_2-norm-based regularization procedure, there are a large number of nonzero, small weight values after regularization. Due to the nonconvex property in ADMM regularization, our observation is that removing these weights can maintain the accuracy or even slightly improve the accuracy occasionally. As a result, we define two thresholds, a column-wise threshold and a filterwise threshold, for each DNN layer. When the L_2 norm of a column (or filter) of weights is below the threshold, the column (or filter) will be removed. Also, the corresponding channel in layer $i+1$ can be removed upon filter removal in layer i. Structures in each DNN layer will be maintained after this purification step.

VI. Discussion

A. Discussion on Hyperparameter Determination and Initialization

A very critical question is how to determine the hyperparameters, in a highly efficient and reliable manner. We need to determine both the target overall pruning rate and specific pruning rate for each layer, both required in the ADMM-based solution. On AlexNet, the pruning rate in each layer for nonstructured pruning and structured pruning is shown in the prior work iterative pruning [7] and SSL [10], respectively. A simple but effective hyperparameter determination method is as follows. We set the target overall pruning rate in the first ADMM-based weight pruning step to be around 1.5× compared with what can be achieved (without accuracy loss) in prior work. The target overall pruning rate in the second step will be doubled compared with the first step or even further increased if there is still a margin of improvement. The per-layer pruning rate will be inherited from the results in prior work and increased proportionally. According to our experiments, the above heuristic generates consistently higher pruning rates than prior work without accuracy loss. We also test the sensitivity of the pruning rate in each layer when the total pruning rate (or sparsity) is fixed. When the total column sparsity for conv2-5 on AlexNet is 79.2%, we try a unified sparsity (79.2% for every layer in conv2-5) and different sparsity (details will appear in Table I), the experiment for unified sparsity only has 0.1% more accuracy loss, which means that when the total pruning rate (or sparsity) is fixed, the distribution of pruning rate in each layer is not sensitive on AlexNet, a unified sparsity also works well.

For other DNNs such as ResNet-18 and ResNet-50, we do not have a prior knowledge of the pruning rate in each layer, and it is complicated to decide different pruning rates in each layer since there are lots of layers in these DNNs. Thus, we use a unified pruning rate for all the layers, we start from 2× pruning rate and progressively increase it in every step. Our experiment results demonstrate that we achieve a higher overall pruning rate compared with prior work by using a unified pruning rate for all the layers. For people without expert knowledge on a DNN, we recommend them first set a small unified pruning rate (such as 2×) for all the layers and then use the progressive pruning method to increase the pruning rate until they observe accuracy loss.

For the hyperparameters α_i, β_i, θ_i, and γ_i discussed in Section IV-C, they are derived by the shape and desired sparsity of the ith layer. For example, in filterwise structured pruning, α_i equals the number of filters in the ith layer that multiplies the desired sparsity in the ith layer.

For nonconvex problems in general, there is no guarantee that ADMM will converge to an optimal point. ADMM can converge to different points for different choices of initial values $\{\mathbf{Z}_i^0\}$ and $\{\mathbf{U}_i^0\}$ and penalty parameters $\{\rho_i\}$ [26]. To resolve this limitation, we set the pretrained model $\{\mathbf{W}_i^p, \mathbf{b}_i^p\}$, a good solution of $\min_{\{\mathbf{W}_i\},\{\mathbf{b}_i\}} f(\{\mathbf{W}_i\},\{\mathbf{b}_i\})$, to be the starting point when we use SGD to solve problem (8). We initialize $\mathbf{Z}_i^0$ by keeping the l_i elements of $\mathbf{W}_i^p$ with the largest magnitude and set the rest to be zero. We set $\mathbf{U}_1^0 = \cdots = \mathbf{U}_N^0 = 0$. For problem (8), if the penalty parameters $\{\rho_i\}$ are too small, the solution will be close to the minimum of $f(\cdot)$ but fail to regularize the weights, and the ADMM procedure may converge slowly or not converge at all. If the penalty parameters are too large, the solution may regularize the weights well but fail to minimize $f(\cdot)$, and therefore, the accuracy of the DNN will be degraded. In actual experiments, We start from smaller ρ_i values, say $\rho_1 = \cdots = \rho_N = 1.5 \times 10^{-3}$, and gradually increase with ADMM iterations. This coincides with the theory of ADMM convergence [58], [59]. It in general takes 8–12 ADMM iterations for convergence, corresponding to 100–150 epochs in PyTorch, a moderate increase compared with the original DNN training.

B. Discussions and Illustration of Effectiveness

The proposed StructADMM is different from the conventional utilization of ADMM, i.e., to accelerate the convergence of an originally convex problem [26], [27]. Rather, we propose to integrate the ADMM framework with SGD. Aside from

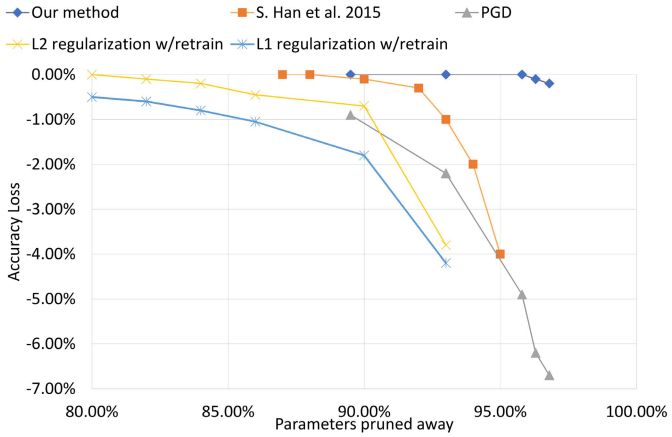


Fig. 5. Comparison of our method and other methods on nonstructured pruning rate of the overall AlexNet model.

recent mathematical optimization results [58], [59] illustrating the advantage of ADMM with combinatorial constraints, the advantage of the proposed StructADMM framework can be explained in the deep learning context as described next.

The proposed StructADMM (8) can be understood as a smart, dynamic L_2 regularization method, in which the regularization target $\mathbf{Z}_i^k - \mathbf{U}_i^k$ will change judiciously and analytically in each iteration. On the other hand, conventional regularization methods (based on L_1, L_2 norms, or their combinations) use a fixed regularization target, and the penalty is applied on all the weights. This will inevitably cause accuracy degradation. In contrast, StructADMM will not penalize the remaining, nonzero weights as long as there are enough weights pruned to zero. For an illustration, Fig. 5 shows the comparison results (Top-5 accuracy loss versus overall pruning rate) on the nonstructured pruning of the whole AlexNet model. Methods to compare include iterative pruning, two regularizations with retraining, and proximal gradient descent (PGD) [60]. We use the same hyperparameters in StructADMM as baseline regularizations and PGD for fairness. It is clear that StructADMM outperforms the others, while regularization-based methods even result in lower performance than [7] because they will penalize all the weights.

We can clearly observe the performance ranking of these techniques from Fig. 5. The proposed progressive framework outperforms all other methods. The second is one-shot ADMM-based pruning. The third is iterative pruning and retraining heuristic. Also, the last is fixed regularizations and PGD. We know that fixed regularizations and PGD suffer from penalizing all weights even if they are not pruned, thereby resulting in notable accuracy degradation.

VII. Experiment Results for Structure Pruning

In this section, we evaluate the proposed StructADMM framework, based on the ImageNet, CIFAR-10, MNIST, and UCF-101 [63] (activity detection) data sets, using the AlexNet [1], VGGNet [5], ResNet-18/ResNet-50 [24], MobileNet-V2 [64], and LeNet-5 [65] DNN models. We focus and compare on the CONV layer structured pruning results, including combined filter (row) and column pruning, and filter-only pruning, for a fair comparison with prior work. In our experiment, the pruning rate is defined based on sparsity.

Due to the compatibility of StructADMM with DNN training, directly training a DNN model using the framework achieves the same result as using a pretrained DNN model. When a pretrained DNN model is utilized, we limit the number of epochs in the single-step ADMM or one single step in the progressive framework to be 120, similar to the original DNN training in PyTorch and much lower than the iterative pruning heuristic. In fact, for ImageNet data set, we adopt a single-step ADMM-based solution. This is to illustrate that StructADMM can outperform prior work even with the similar time of original DNN training. For other data sets, we adopt the progressive solution. This is to illustrate how much StructADMM can outperform prior work given enough training time.

Besides the enhancement in pruning rates compared with prior work, we also aim to investigate the actual inference time speedups in various platforms, as well as the effect of different structured pruning techniques (filter/row pruning and column pruning) and combinations. Will a combination of filter and column pruning schemes run faster than filter pruning alone? To answer this question, we have conducted extensive speedup testings on two GPU nodes, the high-performance NVIDIA GeForce GTX 1080Ti and the low-power NVIDIA Jetson TX2, as well as the Intel I7-6700K Quad-Core CPU. For mobile devices, we also measure the inference time of the pruned models using the latest Qualcomm Adreno 640 GPU in Samsung Galaxy S10 smartphone.

The training of sparse DNN models is performed in PyTorch [66] using NVIDIA 1080Ti, 2080Ti, and Tesla P100 GPUs. The comparisons on GEMM computation efficiency and acceleration use batch size 1, which is typical for inference [7], [10], [14]. The original DNN models and structured sparse models use cuBLAS on GPU and Intel Math Kernel Library (MKL) on CPU. For mobile phone acceleration, our mobile DNN acceleration framework is a compiler-assisted, strong framework by itself. For the original ResNet-50 on ImageNet, it achieves 48-ms inference times, incorporating the Winograd algorithm for acceleration. These results, as starting points, outperform current mobile DNN acceleration frameworks, such as TensorFlow-Lite [67] and TVM [68].

Our models are shared at the link http://bit.ly/2M0V7DO.

A. Experiment Results on Models Using ImageNet Data Set

First, we compare our method with the two configurations of the SSL method [10] on AlexNet/CaffeNet. The first has no accuracy degradation (Top-1 error 42.53%) and average sparsity of 33.3% on conv2–conv5. We note that the 1st CONV layer of AlexNet/CaffeNet is very small with only 35k weights compared with 2.3M in conv2–conv5 and is often not the optimization focus [10], [14]. The second has around 2% accuracy degradation (Top-1 error 44.66%) with total sparsity of 84.4% on conv2–conv5. The first configuration focuses on column sparsity only. The second configuration focuses on a combined row (filter) and column sparsity.

TABLE I

COMPARISON OF OUR METHOD AND SSL METHOD ON COLUMN SPARSITY WITHOUT ACCURACY LOSS ON ALEXNET/CAFFENET MODEL FOR IMAGENET DATA SET

Method	Top-1 Acc Loss	Statistics	conv1	conv2	conv3	conv4	conv5	conv2-5
SSL [10]	0%	column sparsity	0.0%	20.9%	39.7%	39.7%	24.6%	33.3%
		pruning rate	1.0×	1.3×	1.7×	1.7×	1.3×	1.5×
our method	0%	column sparsity	0.0%	70.0%	77.0%	85.0%	81.0%	79.2%
		GPU1×	1.00×	2.27×	3.35×	3.64×	1.04×	2.58×
		GPU2×	1.00×	2.83×	3.92×	4.63×	3.22×	3.65×
		pruning rate	1.0×	3.3×	4.3×	6.7×	5.3×	4.8×

TABLE II

COMPARISON OF OUR METHOD AND SSL METHOD ON COLUMN AND ROW SPARSITY WITH LESS THAN 2% ACCURACY LOSS ON ALEXNET/CAFFENET FOR IMAGENET DATA SET

Method	Top-1 Acc Loss	Statistics	conv1	conv2	conv3	conv4	conv5	conv2-5
SSL [10]	2.0%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%	84.4%§
		row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%	
		pruning rate	1.1×	3.2×	7.7×	12.3×	5.2×	6.4×
our method	0.7%	column sparsity	0.0%	63.9%	78.1%	87.0%	84.9%	86.3%§
		row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%	
		CPU×	1.05×	2.82×	6.63×	10.16×	5.00×	6.16×
		GPU1×	1.00×	1.28×	4.31×	1.75×	1.52×	2.29×
		GPU2×	1.00×	2.34×	6.85×	6.99×	4.15×	5.13×
		pruning rate	1.1×	3.1×	7.3×	14.5×	6.6×	7.3×
our method	2.0%	column sparsity	0.0%	87.5%	90.0%	90.5%	90.7%	93.7%§
		row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%	
		CPU×	1.05×	8.00×	14.68×	14.22×	7.71×	11.93×
		GPU1×	1.00×	2.39×	5.34×	1.92×	2.04×	3.15×
		GPU2×	1.00×	4.92×	12.55×	8.39×	6.02×	8.52×
		pruning rate	1.1×	9.2×	16.8×	19.8×	8.4×	15.0×

§ Total sparsity accounting for both column and row sparsity.

Table I shows the comparison of our method with the first configuration of SSL. We generate a configuration with no accuracy degradation compared with the original model (the original model of our work is higher than that in SSL). We can achieve a much higher degree of sparsity of 79.2% on conv2–conv5. This corresponds to 4.8× weight pruning rate, which is significantly higher compared with 1.5× pruning in conv2–conv5 in [10].

We test the actual GPU accelerations using two GPUs: GPU1 is NVIDIA 1080Ti and GPU2 is NVIDIA TX2. The acceleration rate is computed with respect to the corresponding layer of the original DNN executing on the same GPU and same setup. One can observe that the average acceleration of conv2–conv5 on 1080Ti is 2.58×, whereas the average acceleration on TX2 is 3.65×. These results clearly outperform the GPU acceleration of 49% reported in SSL [10] without accuracy loss, as well as the more recent work [14]. The acceleration rate on TX2 is higher than 1080Ti because the latter has a high parallelism degree, which will not be fully utilized when the matrix size GEMM of a CONV layer is significantly reduced. Another technique cuDNN can use implicit GEMM for better performance than cuBLAS, but we still outperform it on the inference time without accuracy loss. For example, we compare our result in Table I to the baseline results with cuDNN, and our inference time with cuBLAS is also 2.8× lower than the result with baseline cuDNN on original, uncompressed DNN on TX2 GPU.

Table II shows the comparison with the second configuration of SSL. With similar (and slightly higher) sparsity in each layer as SSL, we can achieve less accuracy loss. With the same relative accuracy loss (a moderate accuracy loss within 2% compared with original DNN), a higher degree of 93.7% average sparsity of conv2–conv5 is achieved, translating into 15.0× weight pruning. The actual GPU acceleration results are also high: 3.15× on 1080Ti and 8.52× on TX2. One can clearly see that the speedup on 1080Ti saturates because the high parallelism degree cannot be fully exploited. The acceleration on CPU can be higher under this setup, reaching 11.93× on average on conv2–conv5.

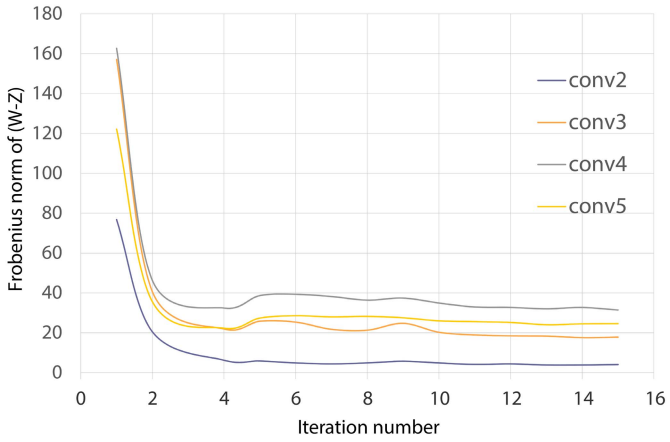


Fig. 6. Convergence behavior of the ADMM regularization procedure for the convolutional layers 2–5 of AlexNet.

Fig. 6 shows the convergence behavior of ADMM regularization (in StructADMM framework, single-step ADMM), using the experiment that achieves 4.8× structured pruning rate on AlexNet without accuracy loss. We can observe that ADMM regularization converges in around 12 iterations. The number of ADMM iterations is generally 9–12 for most of the

TABLE III

STRUCTURED PRUNING RESULTS ON RESNET-18 MODEL FOR IMAGENET DATA SET

Method	Prune rate	Top 5 accuracy
Original	1.0×	89.0%
DCP [47]	1.5×	88.9%
DCP [47]	2.0×	87.6%
Our method (column prune)	2.5×	89.0%
Our method (filter + column prune)	3.0×	88.7%

TABLE IV

STRUCTURED PRUNING RESULTS ON RESNET-50 MODEL FOR IMAGENET DATA SET

Method	Prune rate	Top 5 accuracy
Original	1.0×	92.7%
ThiNet-50 [61]	2.0×	90.0%
ThiNet-30 [61]	3.3×	88.3%
NISP [44]	1.8×	90.2%
Efficient ConvNet [45]	1.4×	91.1%
DCP [47]	2.0×	92.3%
Our method (filter prune)	2.7×	91.9%
Our method (filter + column prune)	2.7×	92.3%

test cases. In each iteration, we need around 10%–20% of the number of epochs as original DNN training.

In addition, in Tables III and IV, we demonstrate the structured (filter or column) pruning results on ResNet-18 and ResNet-50 models for ImageNet data set. We compare with a list of prior work, and these prior work focus on filter pruning only (we do not find prior work on column pruning on these two models). As shown in the tables, we achieve simultaneously a higher pruning rate (weight parameter reduction rate) and higher accuracy compared with prior work when only applying filter (row) pruning. Also, we can observe that column pruning results in a higher pruning rate and/or higher accuracy compared with filter pruning. This is because of the higher flexibility in column pruning by modifying filter shapes. For large-scale GPU/CPU acceleration when GEMM computation is utilized, column pruning (and potentially effective combination with filter pruning) will be more effective than filter pruning only.

B. Experiment Results on Models Using CIFAR-10 Data Set

Tables V–VII show the structured pruning results on ResNet-18, VGG-16, and MobileNet-V2 models, respectively, on the CIFAR-10 data set. We compare with a list of prior work (these prior work focus on filter pruning only). As a two-step, progressive ADMM framework is utilized, we achieve a higher performance gain compared with prior work. As our filter pruning already outperforms prior work, a further increase in pruning rate can be achieved through a combination of filter and column pruning. With a proper combination, a significant improvement over prior work can be achieved. As shown in Table V, the structured pruning rate 54.2× is 32× higher compared with prior work AMC [46] and NISP [44] or over 100× higher considering the difference of the original DNN (these prior work use a larger DNN ResNet-50 or ResNet-56), with even higher accuracy.

For Tables V and VI, the amount of FLOPs reduction and measured mobile inference time (using Adreno 640 mobile GPU with OpenCL code generation from a custom-optimized compiler) are also presented. We can observe that the amount of FLOPs reduction is also highly effective using the proposed StructADMM framework. Besides, the end-to-end mobile inference time is significantly reduced to even beyond real-time requirement. We can also observe the benefit of combined filter and column pruning (than filter pruning only) in mobile execution acceleration, although the inference acceleration rate is not exactly proportional to FLOPs reduction. The latter is because of I/O overheads and the Winograd algorithm applied to the original, unpruned neural network.

TABLE V

STRUCTURED PRUNING RESULTS (ALONG WITH END-TO-END MOBILE INFERENCE TIME) ON RESNET-18 (RESNET-50 IN PRIOR WORK AMC AND RESNET-56 IN PRIOR WORK NISP) FOR CIFAR-10 DATA SET

Method	Prune rate	Top 1 accuracy
Original	1.0×	93.9%
AMC [46]	1.7×	93.5%
NISP [44]	1.7×	93.2%
Our method (filter prune)	8.0×	93.9%
Our method (filter + column prune)	54.2×	93.8%
Flops reduction	**1:** 4.7×	**2:** 12.2×
Inference time (original: $11ms$)	**1:** $2.4ms$	**2:** $1.5ms$

TABLE VI

STRUCTURED PRUNING RESULTS (ALONG WITH END-TO-END MOBILE INFERENCE TIME) ON VGG-16 MODEL FOR CIFAR-10 DATA SET

Method	Prune rate	Top 1 accuracy
Original	1.0×	93.7%
2PFPCE [62]	4.0×	92.8%
2PFPCE [62]	8.3×	91.0%
Efficient ConvNet [45]	2.7×	93.4%
FPGM [22]	1.6×	93.2%
Our method (filter prune)	13.7×	93.2%
Our method (filter + column prune)	50.0×	93.1%
Flops reduction	**1:** 3.1×	**2:** 8.4×
Inference time (original: $14ms$)	**1:** $4.8ms$	**2:** $2.6ms$

Recent work [69] points out an intriguing aspect of structured weight pruning. When one trains from scratch based on the structure (without weight values) of a pruned model, we can often retrieve the same accuracy as the model after pruning. We incorporate this "training from scratch" process based on the results of filter pruning and combined filter and column pruning and derive interesting results. When this process is performed based on the result of filter pruning, it can recover the similar accuracy. The insight is that filter pruning is similar to finding a smaller DNN model, and in this case, the main merit of StructADMM framework is to discover such DNN model (our method still outperforms prior work in this case). On the other hand, when this process is performed based on the result of combined structured pruning, the accuracy cannot be recovered. For example, the accuracy is only 91.9% based on the ResNet-18 pruning results, while the combined structured pruning yields 93.8% accuracy. The underlying insight is that the combined pruning is not just training a smaller DNN model, but with adjustments of filter/kernel shapes. In this case, the pruned model represents a solution that cannot be achieved through performing DNN training only, even with detailed structures already given. In this case, the StructADMM framework will be more valuable due to the importance of pruning from a full-sized DNN model as a starting point.

TABLE VII

STRUCTURED PRUNING RESULTS ON MOBILENET-V2 MODEL FOR CIFAR-10 DATA SET

Method	Prune rate	Top 1 accuracy
Original	1.0×	95.1%
DCP [47]	1.4×	94.7%
Our method (filter prune)	4.7×	95.3%
Our method (filter prune)	4.9×	95.1%
Flops reduction	**1:** 3.3×	**2:** 3.9×
Inference time (original: 4.0ms)	**1:** 1.3ms	**2:** 1.3ms

TABLE VIII

COMPARISON OF OUR METHOD AND SSL METHOD ON FILTERWISE AND CHANNELWISE STRUCTURED SPARSITY ON THE LENET-5 MODEL FOR MNIST DATA SET

LeNet #	Error	Filter # §	Channel # §
1 (Original)	0.87%	20—50	1—20
2 (SSL)	0.80%	5—19	1—4
3 (Our method)	0.77%	5—19	1—4
4 (SSL)	1.00%	3—12	1—3
5 (Our method)	0.79%	3—12	1—3

§ In the order of conv1—conv2.

TABLE IX

COMPARISON OF OUR PROPOSED METHOD AND THE SSL METHOD ON SHAPEWISE STRUCTURED SPARSITY ON THE LENET-5 MODEL FOR MNIST DATA SET

LeNet #	Error	Filter size # §	Channel # §
1 (Original)	0.87%	20—50	1—20
6 (SSL)	0.80%	21—41	1—2
7 (Our method)	0.69%	21—41	1—2
8 (SSL)	1.00%	7—14	1—1
9 (Our method)	0.90%	7—14	1—1

§ In the order of conv1—conv2.

TABLE X

STRUCTURED PRUNING RESULTS ON THE 3-D RESNET-18 MODEL FOR UCF-101 DATA SET

Method	Prune rate / Top-1 accuracy loss
Original	1× / 0%
Direct masked mapping	2.5× / 2.6%
PGD	2.5× / 2.0%
Our method	2.5× / 0.3%

C. Experiment Results on LeNet-5 Model for MNIST Data Set

For the LeNet-5 model, we implement our experiments for MNIST data set. Table VIII shows the comparison of our proposed method and SSL method on filterwise and channelwise structured sparsity, and Table IX shows the comparison on shape-wise structured sparsity. In both cases, our proposed method can achieve higher test accuracy on the same pruning rate compared with the SSL method. The average accuracy gain of our method is above 0.1%, which is notable in the MNIST data set.

D. Experiment Results on ResNet Model for UCF-101 Data Set

In order to demonstrate the broader application of our proposed method, we implement our method on (3D-convolution) ResNet-18 [70] for UCF-101 data set [63] (for activity detection). We compare with the PGD method. The results are shown in Table X, and we achieve higher accuracy compared with PGD on the same pruning rate.

TABLE XI

COMPARISONS OF WEIGHT PRUNING RESULTS ON THE ALEXNET MODEL FOR IMAGENET DATA SET

Method	Top-5 Acc.	No. Para.	Rate
Uncompressed	80.3%	61.0M	1×
Network Pruning [7]	80.3%	6.7M	9×
Optimal Brain Surgeon [19]	80.0%	6.7M	9.1×
Low Rank and Sparse Decomposition [71]	80.3%	6.1M	10×
Fine-Grained Pruning [29]	80.4%	5.1M	11.9×
NeST [9]	80.2%	3.9M	15.7×
Dynamic Surgery [18]	80.0%	3.4M	17.7×
Single-step ADMM [28]	80.2%	2.9M	21×
Hoyer-Square [48]	80.2%	2.86M	21.3×
Progressive Weight Pruning	80.2%	2.02M	30×
Progressive Weight Pruning	80.0%	1.97M	31×

TABLE XII

COMPARISONS OF WEIGHT PRUNING RESULTS ON THE VGG-16 MODEL FOR IMAGENET DATA SET

Method	Top-5 Acc.	No. Para.	Rate
Uncompressed	88.7%	138M	1×
Network Pruning [7]	89.1%	10.6M	13×
Optimal Brain Surgeon [19]	89.0%	10.3M	13.3×
Low Rank and Sparse Decomposition [71]	89.1%	9.2M	15×
Progressive Weight Pruning	88.7%	4.6M	30×
Progressive Weight Pruning	88.2%	4.1M	34×

VIII. EXPERIMENT RESULTS FOR NONSTRUCTURED PRUNING

We evaluate our method for nonstructured pruning on AlexNet and VGG-16 for ImageNet data set. Since we propose a progressive, multistep weight pruning framework, we achieve a higher pruning rate than the single-step ADMM method in the early version of our work [28] on AlexNet. We also achieve a much higher pruning rate compared with other prior work.

Table XI presents the weight pruning comparison results on the AlexNet model between our proposed method and prior work. Our weight pruning results clearly outperform the prior work, in which we can achieve 31× weight reduction rate without loss of accuracy. Our progressive weight pruning also outperforms the single-step ADMM weight pruning in [28] that achieves 21× compression rate.

Table XII presents the comparison results on VGG-16. These weight pruning results we achieved clearly outperform the prior work, consistently achieving the highest sparsity in the benchmark DNN models. On the VGG-16 model, we achieve 30× weight pruning with comparable accuracy with prior work, while the highest pruning rate in prior work is 19.5×. We also achieve 34× weight pruning with minor accuracy loss.

In summary, the experimental results demonstrate that our framework applies to a broad set of representative DNN models and consistently outperforms the prior work. It also applies to the DNN models that consist of mainly convolutional layers, which are different from weight pruning for prior methods. These promising results will significantly contribute to the energy-efficient implementation of DNNs in mobile and embedded systems and on various hardware platforms.

IX. Conclusion

In this article, we proposed a unified, systematic framework of structured weight pruning for DNNs. It is a unified framework for different types of structured sparsity, such as filter-wise, channelwise, shapewise sparsity, as well as nonstructured sparsity. By incorporating SGD with ADMM, our framework updates regularization target analytically in each iteration. Based on ADMM, we further propose a progressive weight pruning framework and a network purification and unused path removal procedure, in order to achieve higher pruning rate without accuracy loss. In our experiments, we achieve 2.58× and 3.65× measured speedup on two GPUs without accuracy loss. The speedups reach 3.18× and 8.52× on GPUs and 10.5× on CPU when allowing a moderate accuracy loss of 2% and reaches 7.6× on the Adreno 640 mobile GPU. Our pruning rate and speedup clearly outperform prior work.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[3] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 30–42, Jan. 2012.

[4] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: https://arxiv.org/abs/1409.1556

[6] K. Makantasis, K. Karantzalos, A. Doulamis, and N. Doulamis, "Deep supervised learning for hyperspectral data classification through convolutional neural networks," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2015, pp. 4959–4962.

[7] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent.*, 2016.

[9] X. Dai, H. Yin, and N. K. Jha, "NeST: A neural network synthesis tool based on a grow-and-prune paradigm," *IEEE Trans. Comput.*, vol. 68, no. 10, pp. 1487–1497, Oct. 2019.

[10] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[11] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5687–5695.

[12] M. Denil *et al.*, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.

[13] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 2014, pp. 1269–1277.

[14] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li, "Coordinating filters for faster deep neural networks," 2017, *arXiv:1703.09746*. [Online]. Available: https://arxiv.org/abs/1703.09746

[15] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 2857–2865.

[16] V. Sindhwani, T. Sainath, and S. Kumar, "Structured transforms for small-footprint deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3088–3096.

[17] L. Zhao, S. Liao, Y. Wang, Z. Li, J. Tang, and B. Yuan, "Theoretical properties for neural networks with weight matrices of low displacement rank," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 4082–4090.

[18] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Adv. Neural Inf. Process. Syst.*, 2016, pp. 1379–1387.

[19] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4860–4874.

[20] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proc. 34th Int. Conf. Mach. Learn.*, Volume 70, 2017, pp. 2498–2507.

[21] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2018, *arXiv:1803.03635*. [Online]. Available: http://arxiv.org/abs/1803.03635

[22] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4340–4349.

[23] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," 2017, *arXiv:1712.01312*. [Online]. Available: http://arxiv.org/abs/1712.01312

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[25] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1389 –1397.

[26] S. Boyd, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2010.

[27] M. Hong and Z.-Q. Luo, "On the linear convergence of the alternating direction method of multipliers," *Math. Program.*, vol. 162, nos. 1–2, pp. 165–199, Mar. 2017.

[28] T. Zhang *et al.*, "A systematic DNN weight pruning framework using alternating direction method of multipliers," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 184–199.

[29] H. Mao *et al.*, "Exploring the regularity of sparse structure in convolutional neural networks," 2017, *arXiv:1705.08922*. [Online]. Available: http://arxiv.org/abs/1705.08922

[30] E. Park, J. Ahn, and S. Yoo, "Weighted-entropy-based quantization for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5456–5464.

[31] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," 2017, *arXiv:1702.03044*. [Online]. Available: http://arxiv.org/abs/1702.03044

[32] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2849–2858.

[33] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4820–4828.

[34] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* New York, NY, USA: Springer, 2016, pp. 525–542.

[35] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[36] A. Ren *et al.*, "ADMM-NN: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 925–938.

[37] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[38] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with ADMM," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3466–3473.

[39] C. Bucilua, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2006, pp. 535–541.

[40] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*. [Online]. Available: http://arxiv.org/abs/1503.02531

[41] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, "Be your own teacher: Improve the performance of convolutional neural networks via self distillation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3713–3722.

[42] L. Zhang, Z. Tan, J. Song, J. Chen, C. Bao, and K. Ma, "SCAN: A scalable neural networks framework towards compact and efficient models," in *Proc. NeurIPS*, 2019, pp. 4027–4036.

[43] C. Ding *et al.*, "CirCNN: Accelerating and compressing deep neural networks using block-circulant weight matrices," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2017, pp. 395–408.

[44] R. Yu *et al.*, "NISP: Pruning networks using neuron importance score propagation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9194–9203.

[45] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Peter Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*. [Online]. Available: http://arxiv.org/abs/1608.08710

[46] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. ECCV*. New York, NY, USA: Springer, 2018, pp. 815–832.

[47] Z. Zhuang *et al.*, "Discrimination-aware channel pruning for deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 875–886.

[48] H. Yang, W. Wen, and H. Li, "Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures," in *Proc. Int. Conf. Learn. Represent.*, 2020. [Online]. Available: https://openreview.net/forum?id=rylBK34FDS

[49] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal SGD for pruning very deep convolutional networks with complicated structure," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4943–4953.

[50] T. Li, B. Wu, Y. Yang, Y. Fan, Y. Zhang, and W. Liu, "Compressing convolutional neural networks via factorized convolutional filters," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 3977–3986.

[51] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad, "A simple effective heuristic for embedded mixed-integer quadratic programming," *Int. J. Control*, vol. 93, no. 1, pp. 2–12, 2017.

[52] C. Leng, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with ADMM," 2017, *arXiv:1707.09870*. [Online]. Available: http://arxiv.org/abs/1707.09870

[53] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, 2013, vol. 30, no. 1, p. 3.

[54] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. COMPSTAT*. New York, NY, USA: Springer, 2010, pp. 177–186.

[55] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[56] S. Liu, M. Fardad, E. Masazade, and P. K. Varshney, "On optimal periodic sensor scheduling for field estimation in wireless sensor networks," in *Proc. IEEE Global Conf. Signal Inf. Process.*, Dec. 2013, pp. 137–140.

[57] S. Chetlur *et al.*, "CuDNN: Efficient primitives for deep learning," 2014, *arXiv:1410.0759*. [Online]. Available: http://arxiv.org/abs/1410.0759

[58] M. Hong, Z.-Q. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," *SIAM J. Optim.*, vol. 26, no. 1, pp. 337–364, 2016.

[59] S. Liu, J. Chen, P.-Y. Chen, and A. Hero, "Zeroth-order online alternating direction method of multipliers: Convergence analysis and applications," in *Proc. Int. Conf. Artif. Intell. Statist.*, vol. 2018, pp. 288–297.

[60] N. Parikh *et al.*, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, pp. 127–239, 2014.

[61] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5058–5066.

[62] C. Min, A. Wang, Y. Chen, W. Xu, and X. Chen, "2PFPCE: Two-phase filter pruning based on conditional entropy," 2018, *arXiv:1809.02220*. [Online]. Available: http://arxiv.org/abs/1809.02220

[63] K. Soomro, A. Roshan Zamir, and M. Shah, "UCF101: A dataset of 101 human actions classes from videos in the wild," 2012, *arXiv:1212.0402*. [Online]. Available: http://arxiv.org/abs/1212.0402

[64] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[65] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[66] A. Paszke *et al.*, "Automatic differentiation in pytorch," in *Proc. Int. Conf. Learn. Represent. Neural Inf. Process. Syst. Workshop*, 2017.

[67] [Online]. Available: https://www.tensorflow.org/mobile/tflite/

[68] T. Chen *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. OSDI*, 2018, pp. 578–594.

[69] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2018, *arXiv:1810.05270*. [Online]. Available: http://arxiv.org/abs/1810.05270

[70] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," 2014, *arXiv:1412.0767*. [Online]. Available: http://arxiv.org/abs/1412.0767

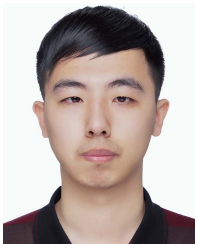
[71] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7370–7379.

Tianyun Zhang received the B.E. degree in optoelectronic information engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2015. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, USA.

His research interests include model compression and efficient implementation for deep neural networks, adversarial robustness on artificial intelligence, and convex and nonconvex optimization.

Shaokai Ye received the B.S. degree in computer engineering from Saint Louis University, St. Louis, MO, USA, in 2015, and the M.S. degree in computer engineering from Syracuse University, Syracuse, NY, USA, in 2018, under the supervision of Dr. Y. Wang. He is currently pursuing the Ph.D. degree from the École polytechnique fédérale de Lausanne (EPFL), Lausanne, Switzerland, under the supervision of Dr. M. Mathis.

His research interest is making AI more efficient and robust.



Xiaoyu Feng (Graduate Student Member, IEEE) received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2018, where he is currently pursuing the Ph.D. degree in electronic engineering.

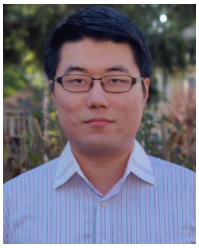
His current research interest lies in neural network compression, domain adaptation, and energy-efficient network architecture design.

Xiaolong Ma received the M.S. degree in electrical engineering from Syracuse University, Syracuse, NY, USA, in 2016. He is currently pursuing the Ph.D. degree in computer engineering with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, USA, supervised by Dr. Yanzhi Wang.

He has published in top conference and journal venues, including the Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI), the European Conference on Computer Vision (ECCV), ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Design Automation Conference (DAC), ACM International Conference on Supercomputing (ICS), International Conference on Parallel Architectures and Compilation Techniques (PACT), and IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS. His research interests include machine learning algorithms, deep learning system design, high-performance computing, and computer vision.

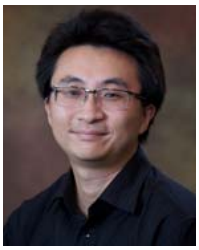
Kaiqi Zhang received the B.E. degree in vehicle engineering from Tsinghua University, Beijing, China, in 2011, and the M.S. degree in computer science from Syracuse University, Syracuse, NY, USA, in 2018.

He is currently working as a Software Engineer with Microsoft Azure, Seattle, WA, USA. His research interests include deep learning, optimization algorithm, GPU acceleration, and cloud computing.



Zhengang Li received the B.E. degree in electronic information engineering from Zhejiang University, Hangzhou, Zhejiang, China, in 2017. He is currently pursuing the Ph.D. degree in computer engineering with Northeastern University, Boston, MA, USA, under the supervision of Prof. Yanzhi Wang.

His current research interests include model compression of deep neural networks, machine learning algorithms, and computer vision.



Jian Tang (Fellow, IEEE) received the Ph.D. degree in computer science from Arizona State University, Tempe, AZ, USA, in 2006.

He is currently a Professor with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, USA. He has published over 150 papers in premier journals and conferences. His research interests include the areas of AI, Internet of Things (IoT), wireless networking, mobile computing, and big data systems.

Dr. Tang is an ACM Distinguished Member. He received the NSF CAREER Award in 2009, several best paper awards, including the 2019 William R. Bennett Prize and the 2019 TCBD (Technical Committee on Big Data) Best Journal Paper Award from IEEE Communications Society, the 2016 Best Vehicular Electronics Paper Award from the IEEE Vehicular Technology Society, and best paper awards from the 2014 IEEE International Conference on Communications and the 2015 IEEE Global Communications Conference. He has served as an Editor for several IEEE journals, including the IEEE TRANSACTIONS ON BIG DATA and the IEEE TRANSACTIONS ON MOBILE COMPUTING. He served as the TPC Co-Chair for a few international conferences, including the IEEE/ACM IWQoS 2019, MobiQuitous 2018, and IEEE iThings 2015, the TPC Vice Chair for the INFOCOM'2019, and the Area TPC Chair for INFOCOM 2017–2018.

Sijia Liu (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Syracuse University, Syracuse, NY, USA, in 2016.

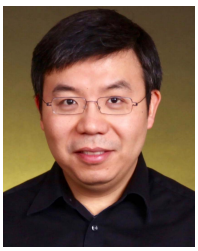
He was a Post-Doctoral Research Fellow with the University of Michigan, Ann Arbor, MI, USA. He is currently a Research Staff Member with the MIT-IBM Watson AI Lab, Cambridge, MA, USA. His research interests include optimization for deep learning and adversarial machine learning.

Dr. Liu received the Best Student Paper Award (Third Place) at ICASSP 2017 and the All-University Doctoral Prize for the Ph.D. degree. He was among the seven finalists of the Best Student Paper Award at Asilomar 2013.

Xue Lin (Member, IEEE) received the bachelor's degree in microelectronics from Tsinghua University, Beijing, China, in 2009, and the Ph.D. degree from the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA, in 2016.

She has been an Assistant Professor with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, USA, since 2017. Her research work has been recognized by several NSF awards and supported by the Air Force Research Lab, Office of Naval Research, Lawrence Livermore National Lab, and DARPA. Her research interests include deep learning security and hardware acceleration, machine learning and computing in cyber-physical systems, high-performance and mobile cloud computing systems, and VLSI.

Dr. Lin received the Best Paper Award at ISVLSI 2014, the Top Paper Award at CLOUD 2014, the Best Technical Poster Award at NDSS 2020, the Spotlight Paper at ECCV 2020, and the First Place at the ISLPED 2020 Design Contest.



Yongpan Liu (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 1999, 2002, and 2007, respectively.

He was a Visiting Scholar with Pennsylvania State University, State College, PA, USA, and the City University of Hong Kong, Hong Kong. He is currently a Professor with the Department of Electronic Engineering, Tsinghua University. He has published over 200 peer-reviewed conference and journal papers and developed several fast sleep/wakeup nonvolatile processors using emerging memory and artificial intelligent accelerators using algorithm-architecture co-optimization. His main research interests include energy-efficient circuits and systems for artificial intelligent, emerging memory devices, and Internet-of-Things (IoT) applications.

Dr. Liu's work has received the Under 40 Young Innovators Award DAC 2017, the Micro Top Pick 2016, the Best Paper Award in ASPDAC 2017 and HPCA 2015, and the Design Contest Awards of ISLPED 2012 and 2013. He served as the General Chair for AWSSS 2016 and IWCR 2018, the Technical Program Chair for NVMSA 2019, and a Program Committee Member for DAC, DATE, ASP-DAC, ISLPED, ICCD, and A-SSCC. He is also an Associate Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS, and *IET Cyber-Physical Systems: Theory & Applications*.

Makan Fardad received the B.S. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 1998, the M.S. degree in electrical engineering from the Iran University of Science and Technology, Tehran, in 2000, and the Ph.D. degree in mechanical engineering from the University of California at Santa Barbara, Santa Barbara, CA, USA, in 2006.

He was a Post-Doctoral Associate with the University of Minnesota, Minneapolis, MN, USA. He joined the Department of Electrical Engineering and Computer Science, Syracuse University. His research interests include modeling, analysis, and optimization of large-scale dynamical networks.

Dr. Fardad was a recipient of the National Science Foundation CAREER Award.

Yanzhi Wang (Member, IEEE) received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2009, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 2014.

His research interests include energy-efficient and high-performance implementations of deep learning and artificial intelligence systems, emerging deep learning algorithms/systems, generative adversarial networks, and deep reinforcement learning.